Day 5 - Testing, Error Handling, and Backend Integration Refinement

Objective:

Day 5 is all about getting our marketplace ready for real-world use. We'll focus on testing every component, optimizing performance, and making sure it can handle live customer traffic. The main priorities include backend integration testing, implementing robust error handling, and fine-tuning the user experience.

Primary feature testing:

1) Product Listing:

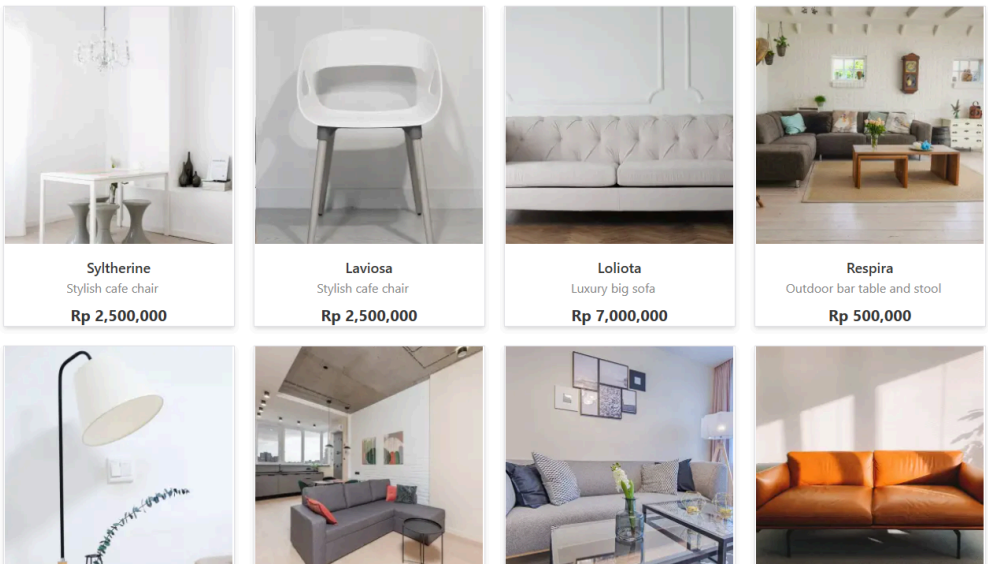
Verify that all the Product are display Correctly as expected in Our MarketPlace

Approaches:

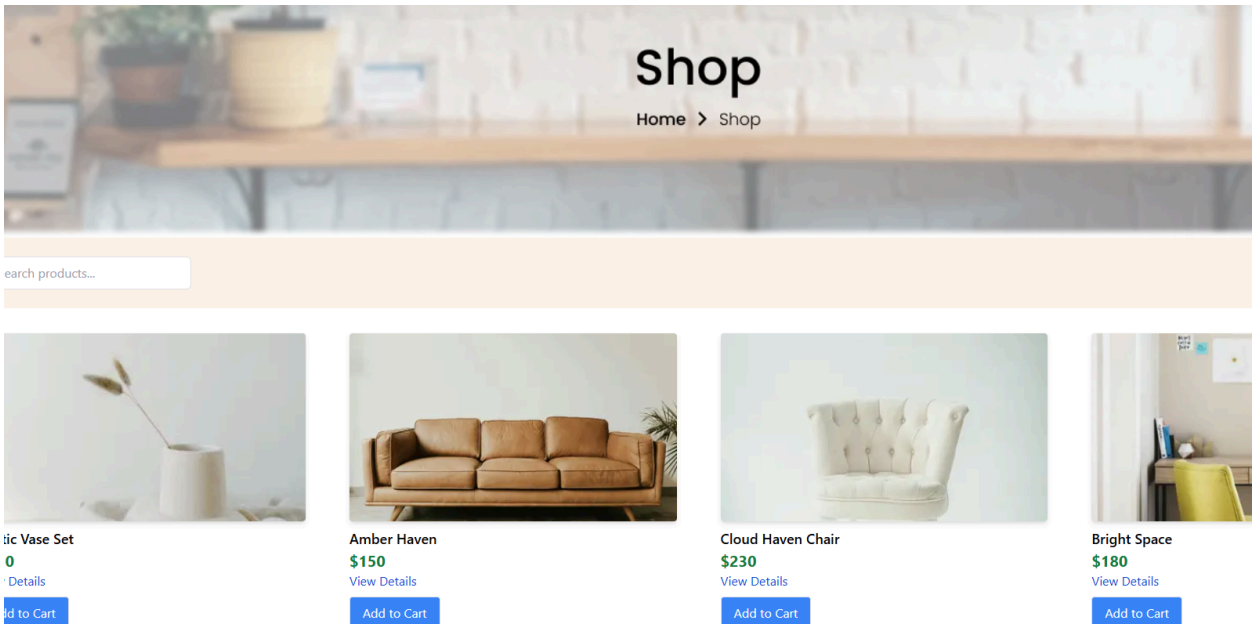
- We first Navigate to Homapage or ProductDisplay Page
- Insure that all the products ,images,prices,tags,and discountpercentage are displayed perfectly
- Insured that all the the product details and description are showing correctly which is fetched from Sanity CMS and Import Properly as expected

Result:

All product images, prices, details, and tags are displayed correctly.



This is my ProductListing Components in which all the Products Shown as Expected according my working



2) Filters And Categories:

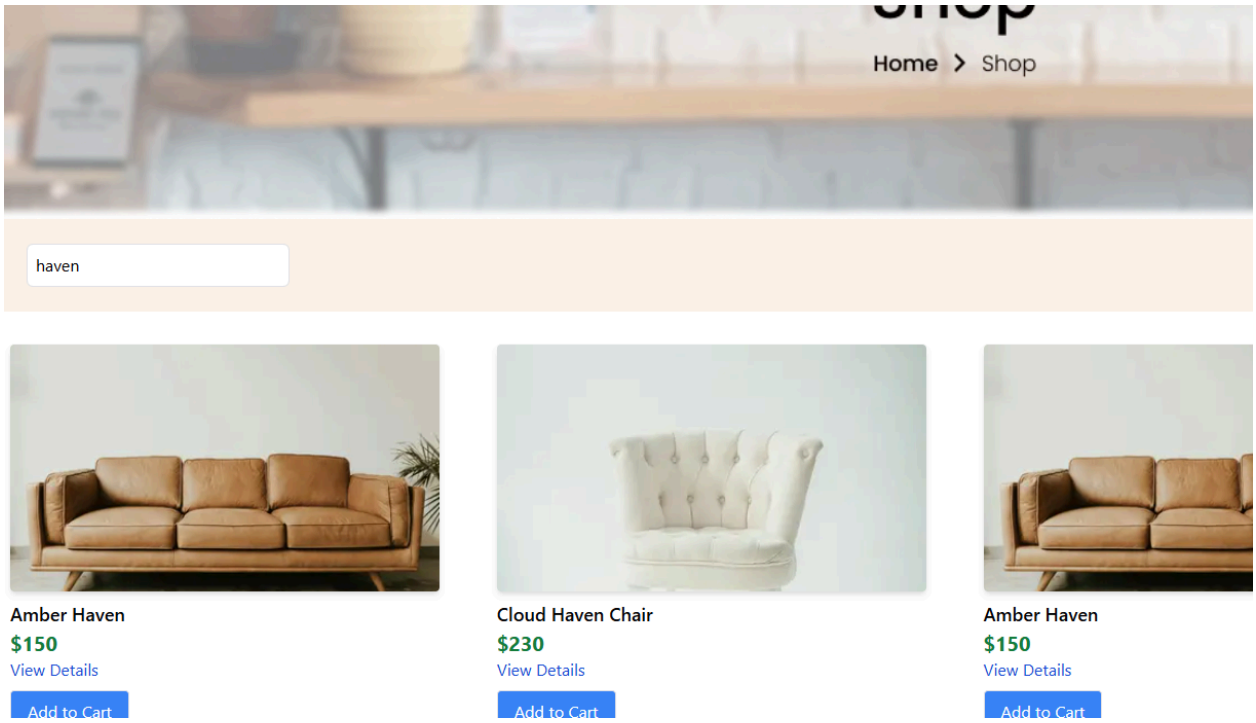
We Verify that my filter and categories components work as we expected in our MarketPlace

Approaches:

- When we navigate towards my Shop Component we first make sure that my product firstly display properly as expected

- Now we check that all the product should display when we search (by category and productname).

Result:



- My Product display properly according to my given Criteria

- Result should indicate that my filter and category components work as expected and the provided criteria when user write any category name it shown proper result according to productname.

3) Cart Component:

Verify that Cart Operate as we expected and according to the criteria (must shown remove , add , and updates)

Approaches:

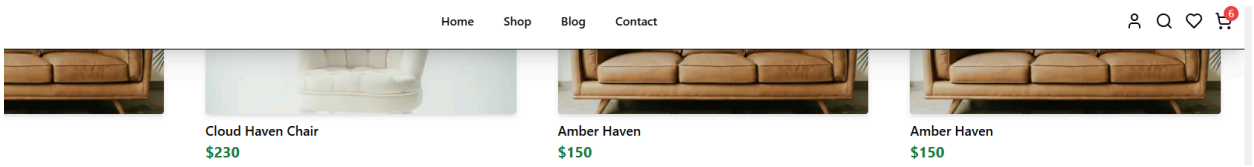
- Check that cart shown correct items which user/customer selected in shop component.

- Ensure that cart shown correct quantity according to selection
- Check that addition and subtraction function in cart components work correctly.
- Verify that Cart Remove options work as expected and Show correct calculation according to the price of selected products and show the proceed option

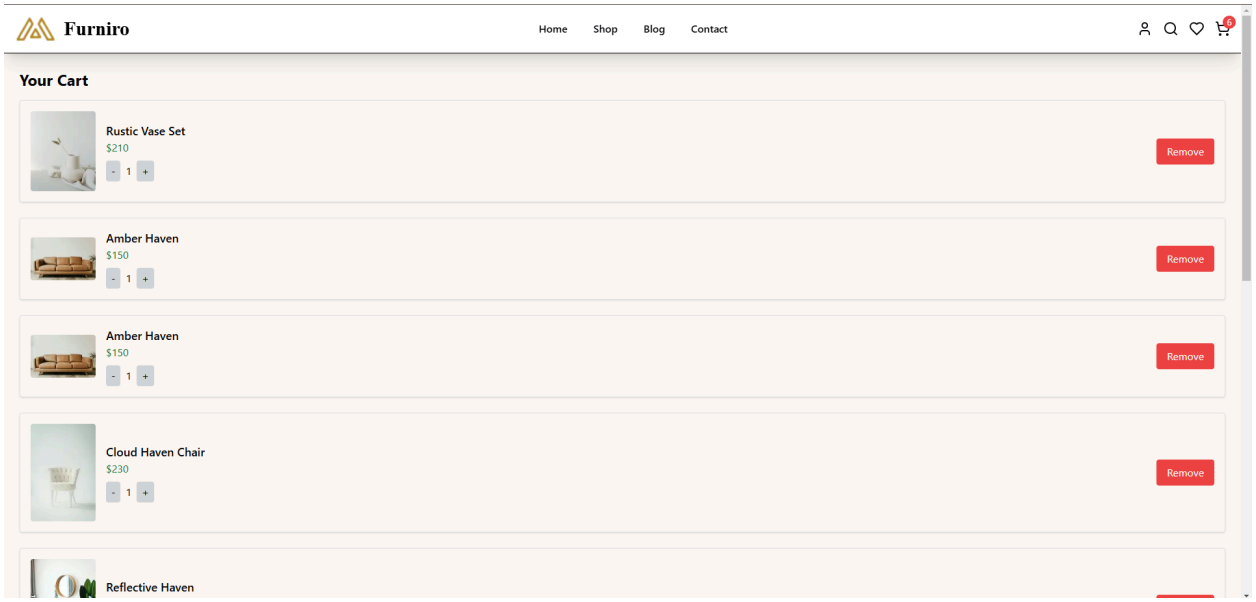
Result:

The cart functionality must operate seamlessly, ensuring an intuitive and efficient user experience. It should:

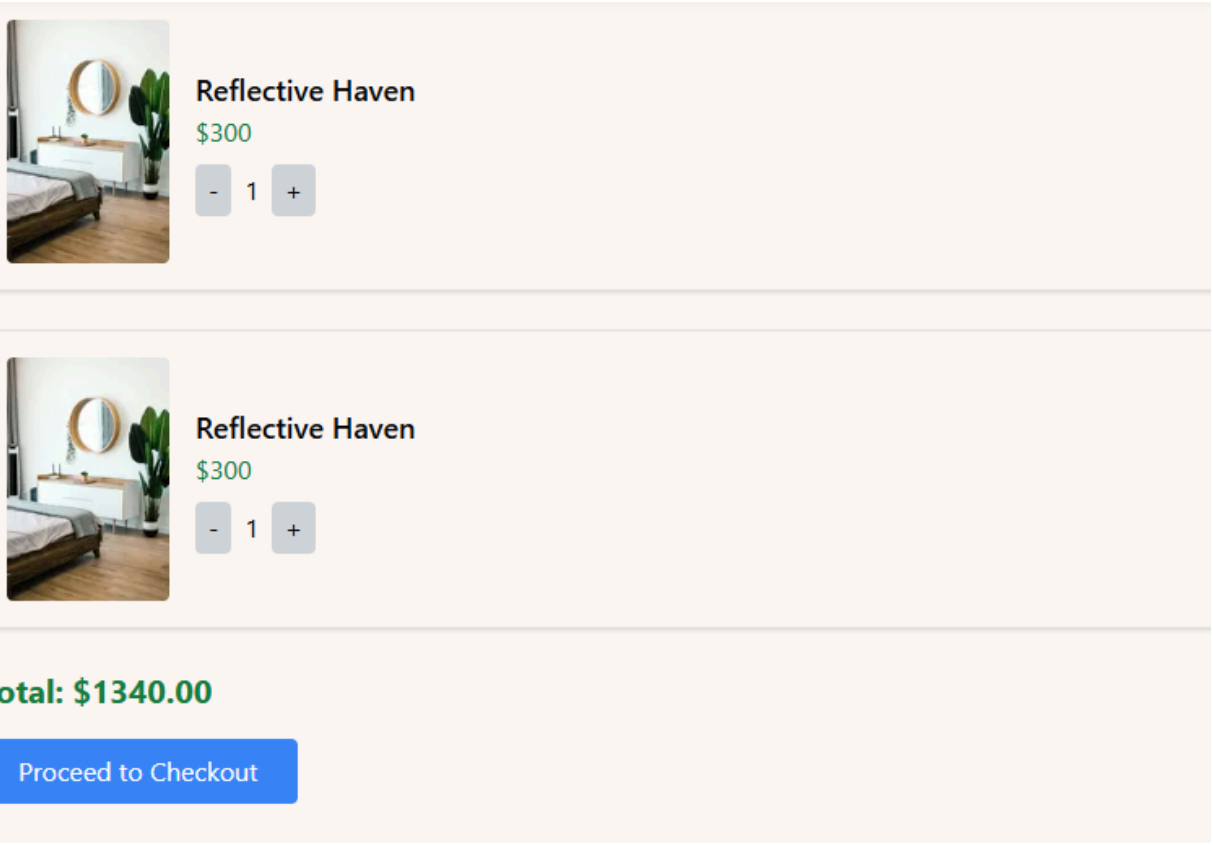
- **Accurately Display Options:** The cart must correctly show options such as **Add, Remove, and Edit** for products
- **Precise Price Calculation:** All price calculations, including item subtotals, discounts, and total amounts, should be accurate and dynamically updated.
- **Correct Product Selection:** The cart should reflect the exact products selected by the user, maintaining accuracy in quantity and variants, if applicable.



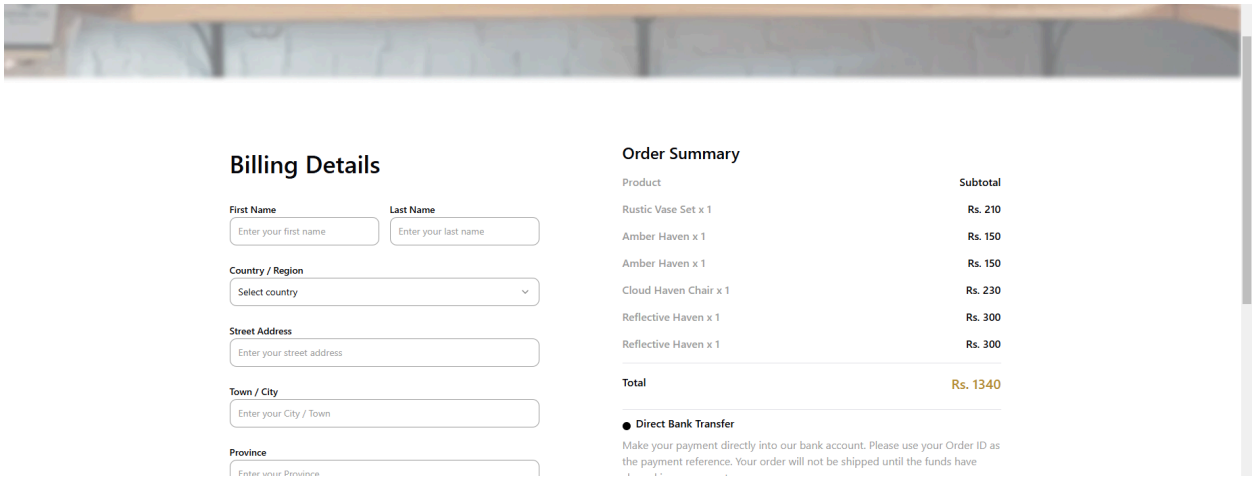
- Product selection component work correctly as shown in my Navbar icon



- Cart Operates Correct working according to Selection of customer



- Calculations display correctly.



4) Checkout Component:

The checkout page must accurately display the selected products, ensuring a seamless and transparent purchasing experience. It should:

- **Accurately Reflect Selections:** All chosen products must be displayed exactly as selected.
- **Correct Pricing & Quantity:** Each item's price, quantity, and total cost should be precisely calculated and shown.
- **Seamless Checkout Experience:** The final order summary should be clear, error-free, and aligned with user expectations.

To complete the purchase, users need to follow these final steps for a seamless checkout experience:

- **Enter Billing Details:** Users must accurately provide their billing information, including name, address, and payment details, ensuring a smooth transaction.
- **Review Order Summary:** The checkout page will display the selected products, including their quantity, price, and total amount, allowing users to verify all details before proceeding.
- **Confirm & Book Order:** Once all details are reviewed, users can finalize their purchase by confirming the order, ensuring a hassle-free and secure booking process.

Select Your Payment Method

Choose a convenient payment option to complete your order securely:

1. Direct Bank Transfer

Make your payment directly to our bank account. Use your **Order ID** as the payment reference. Please note that your order will be processed and shipped only after the funds have been successfully received in our account.

2. Cash on Delivery (COD)

Pay in cash upon delivery. This option allows you to verify your order before making the payment.

Your personal data will be used to enhance your shopping experience, manage your account, and ensure a secure transaction, as outlined in our **Privacy Policy**.

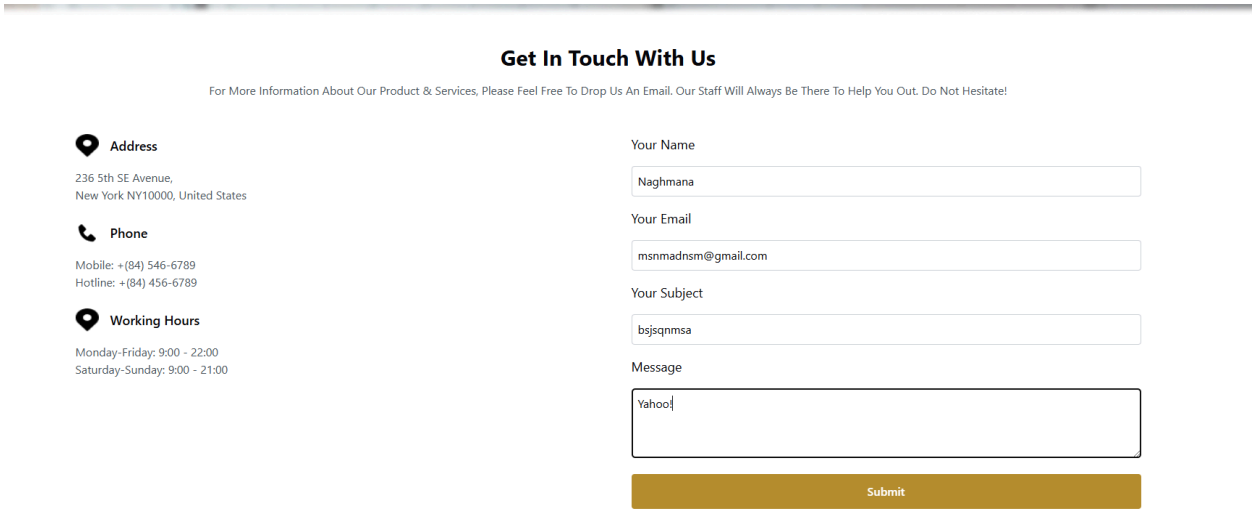
5) Contact & Order Confirmation Process:

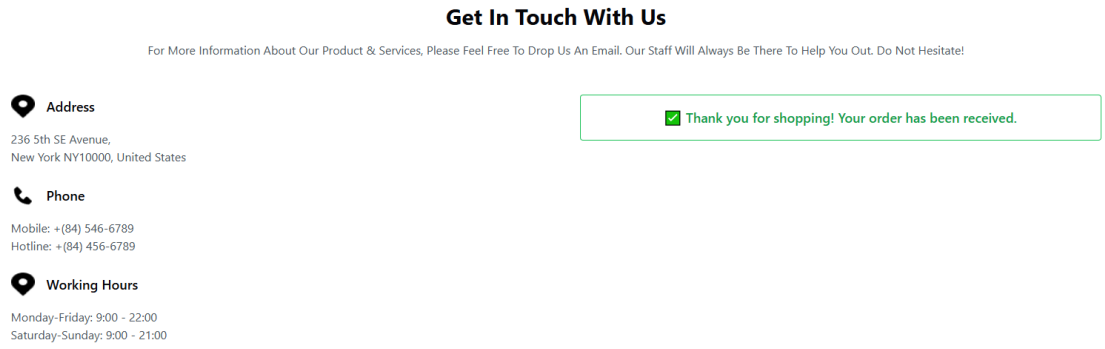
To proceed with your order or inquiries, filling out the **Contact Form** is mandatory. Please ensure the following details are provided:

- **Your Name** – Enter your full name for identification.
- **Your Email** – Provide a valid email address for communication.
- **Subject** – Mention the purpose of your message.
- **Message** – Clearly describe your request or concern.

Without these details, your order cannot be processed.

Once submitted, you will receive a **confirmation message** acknowledging your request, followed by a **thank you message**, ensuring a smooth and transparent communication experience.





6) API Status Report – Apidog Analysis

Endpoint Details

- **API Tool:** Apidog
- **Endpoint:** `https://template6-six.vercel.app/api/products`

- **Status:** Developing
- **Method:** `GET`
- **Last Updated By:** Naghmana Asif
- **Shared Access:** Enabled

Request & Response Overview

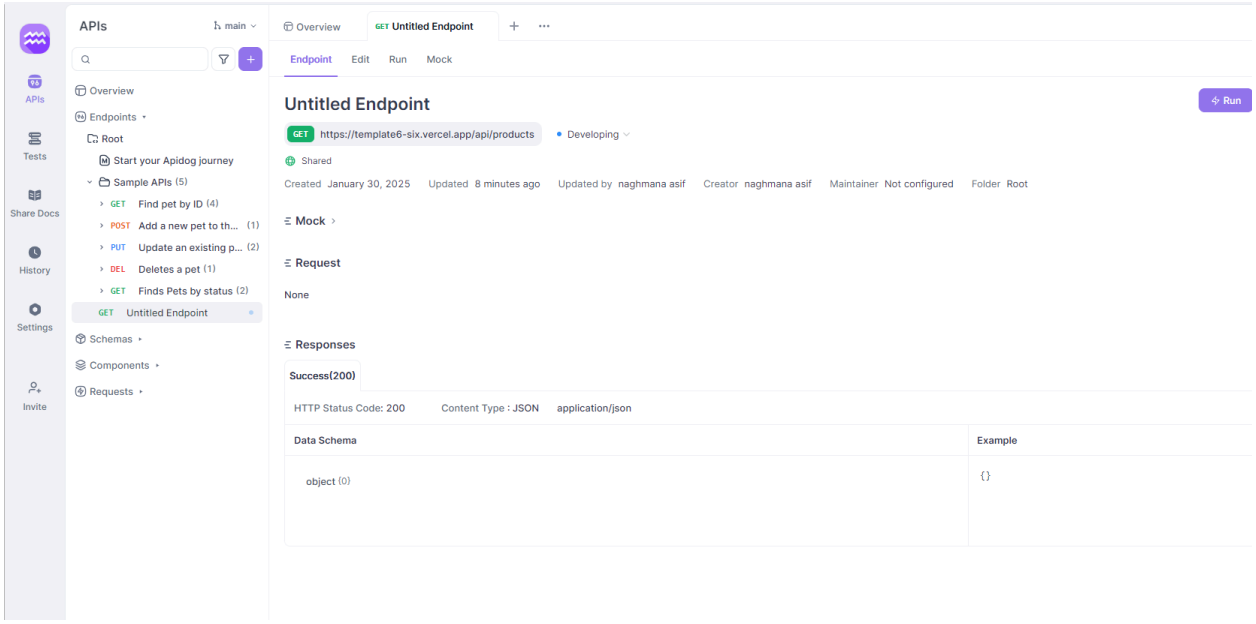
- **Request Parameters:** None
- **HTTP Response Code:** `200 OK` (Success)
- **Content Type:** `application/json`
- **Response Data Schema:** `{}` (Empty Object)

Analysis & Remarks

- The API request successfully returns a **200 OK** response, confirming that it is active.
- However, the response currently returns an **empty object `{}`**, indicating that the expected product data is not yet integrated.
- Further configuration is needed to ensure the correct data retrieval and display.

Conclusion

The API is in its **development stage** and responding correctly, but data population and endpoint refinement are required for full functionality.



7) Lighthouse Website Performance Analysis

Audit Summary

The website was analyzed using **Google Lighthouse**, and the results are as follows:

- **Performance: 70** (Needs Improvement)
- **Accessibility: 85 (Good)**
- **Best Practices: 100 (Excellent)**
- **SEO: 100 (Excellent)**

Key Performance Metrics

- **First Contentful Paint (FCP): 1.0s (Good)**
- **Largest Contentful Paint (LCP): 1.7s (Acceptable)**

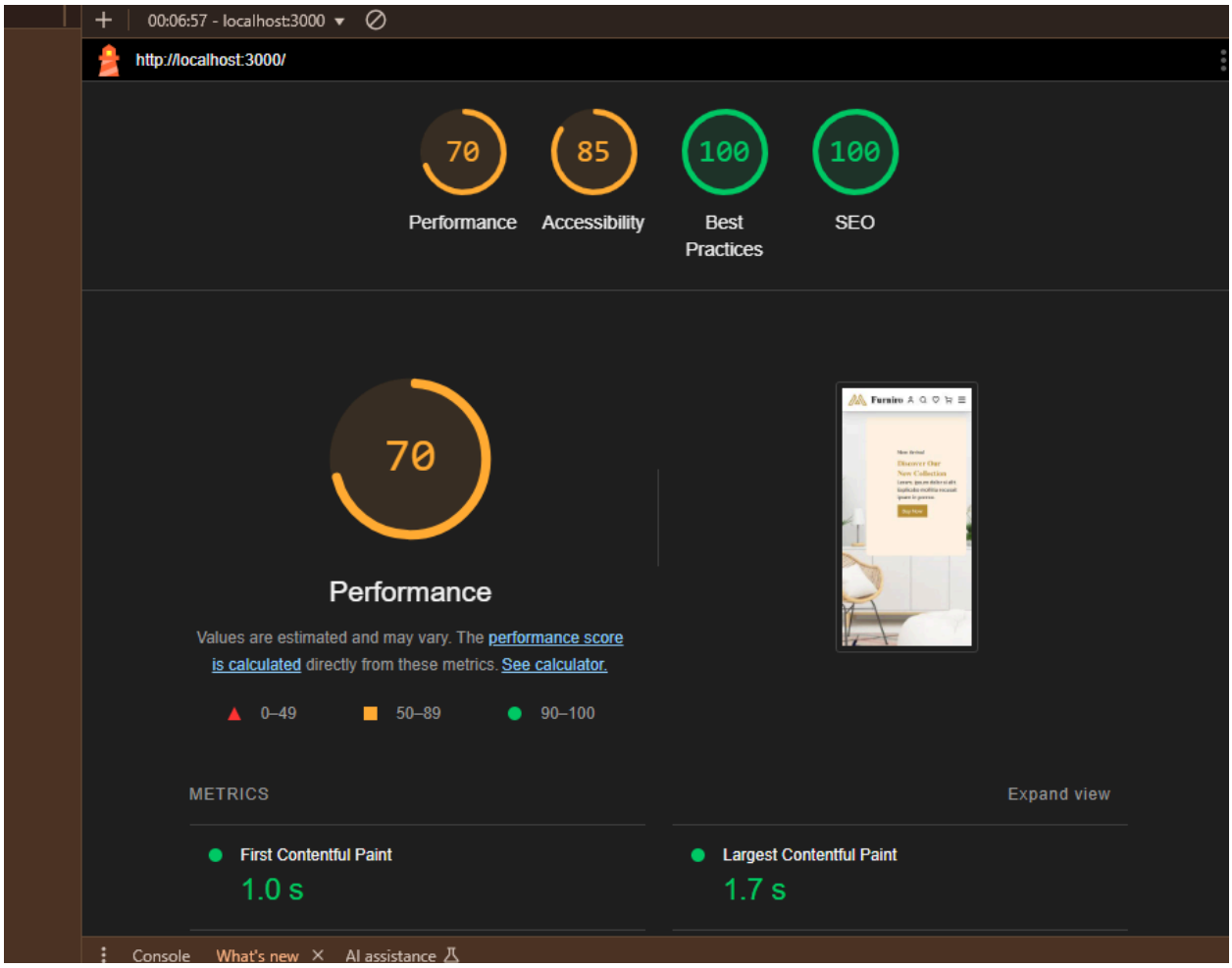
Recommendations for Improvement

1. **Optimize Performance:**
    - Reduce unused JavaScript and CSS.
    - Optimize images and implement lazy loading.

    - Minimize third-party scripts to reduce loading time.
2. **Enhance Accessibility:**
    - Ensure color contrast is sufficient for readability.
    - Add alt text for all images.
    - Improve keyboard navigation and focus states.
3. **Maintain Best Practices & SEO:**
    - Keep security measures in check.
    - Ensure structured data and meta tags are properly configured.

Conclusion

The website performs well in **SEO and best practices** but requires **performance optimization** to improve user experience. Implementing these recommendations will enhance overall speed, accessibility, and efficiency.



8) Product Api Testing:

Test-01:

Api testing by Postman:

Test Summary:

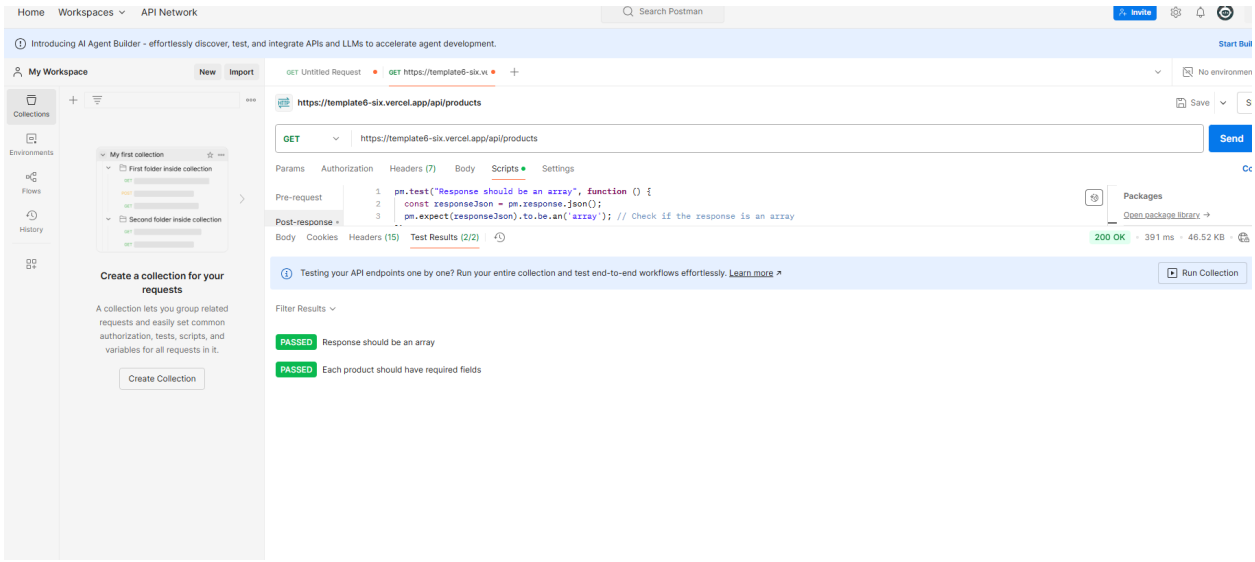
- **Test: API response should be an array.**
- **Result: Passed** (API response is in array format).
- **Test: Each product should have required fields.**
- **Result: Passed** (Each product has all required fields).
- **Response Time: 391 ms.**
- **Response Size: 46.52 KB.**

What Was Tested:

- **API endpoint**: Specify the API endpoint we tested (e.g.https://template6-six.vercel.app/api/products,).
- **Expected Result**: API should return an array, each product should contain required fields.
- **Actual Result**: Postman test results confirming the API returns data in the expected format.

Conclusion:

- Based on the test, the API is working as expected in terms of data format and performance. The response time is under 1 second, and the response contains all required fields for each product.



8) Step-by-Step: Error Handling with try-catch in API Requests

1) **API Request Execution (Try Block):**
   - First, we use the **try block** where we execute our API request. In this case, we are fetching product data from Sanity.
   - If the request is successful, we store the fetched products in the `products` array.

js
CopyEdit

```
try {

  products = await client.fetch(`...`);

}
```

2. **Error Handling (Catch Block):**
   - If the API request fails (due to reasons like a network issue, incorrect URL, etc.), the **catch block** is executed.

- Here, we log the error and display a friendly message to the user, such as "Unable to load products. Please try again later."

js
CopyEdit

```
} catch (error) {

  console.error("Failed to fetch products:", error);

  errorMessage = "Unable to load products. Please try
again later.";

}
```

3. **Handling Different States:**

Error State: If an error occurs (like a network failure or API issue), we show the **error message**.
js
CopyEdit

```
if (errorMessage) {

  return <div>{errorMessage}</div>;

}
```

Loading State: While the products are loading, we display a loading message to keep the user informed that data is being fetched.
js
CopyEdit

```
if (!products.length && !errorMessage) {

  return <div>Loading products...</div>;

}
```

Empty State: If products are fetched but there are no products, we display a "No items found" message.
js
CopyEdit

```
if (products.length === 0) {

  return <div>No items found</div>;

}
```

4. **Successful Data Fetching:**

If everything goes well, we render the `ShopContent` component to display the fetched products.
js
CopyEdit

```
return <ShopContent products={products} />;
```

Benefits of Using Try-Catch:

- **Prevents Crashes**: If an error occurs, the app doesn't crash; we show a user-friendly message instead.
- **User Experience**: Users are informed if something goes wrong, so they can wait or retry.
- **Debugging**: By logging errors to the console, we can easily debug and track down what went wrong.

Code :

```
import { client } from "@/sanity/lib/client";
```

```
import ShopContent from "./ShopContent";



export default async function Shop() {
```

```
let products = [];
let errorMessage = null;



try {
  products = await client.fetch(`
    *[_type == "product"]{
      _id,
      title,
      price,
      category,
      productImage{
        asset->{url}
      }
    }
  `);
} catch (error) {
  console.error("Failed to fetch products:", error);
  errorMessage = "Unable to load products. Please try again later.";
}



if (errorMessage) {
```

```
    return <div>{errorMessage}</div>;
  }



  if (!products.length && !errorMessage) {
    return <div>Loading products...</div>;
  }



  if (products.length === 0) {
    return <div>No items found</div>;
  }



  return <ShopContent products={products} />;
}
```

9) Cross-Browser Testing: Microsoft Edge:

Objective:

To ensure a seamless user experience and consistent rendering across various devices and screen sizes using Microsoft Edge.

Testing Procedure:

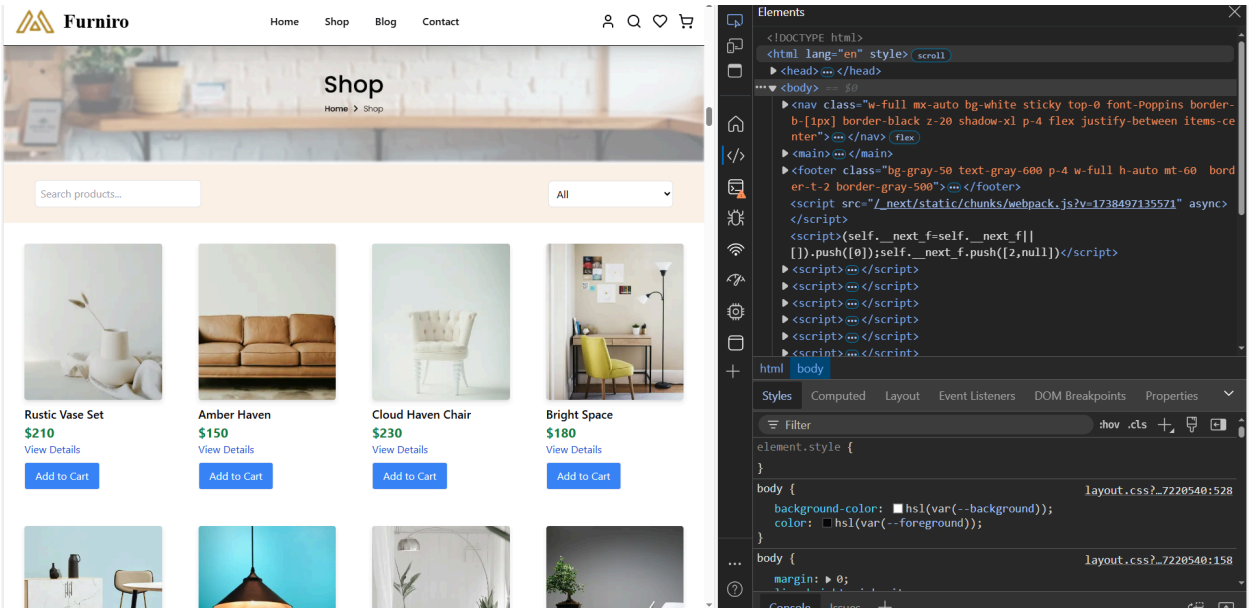
1. **Browser Setup & Tools:**
    - Initiated testing using **Microsoft Edge**, leveraging the built-in **Developer Tools** to simulate various screen sizes and devices.
    - Accessed Developer Tools with the shortcut `F12` or `Ctrl + Shift + I` and enabled **Device Emulation (Responsive Design Mode)** via the device icon or `Ctrl + Shift + M` for a comprehensive testing approach.
2. **Device & Screen Simulation:**
    - Selected from a variety of devices including **mobile phones**, **tablets**, and **desktop** to ensure the layout and functionality remained flawless across all device types.
    - Manually tested on real mobile devices for accurate, on-the-ground feedback beyond simulated environments.
3. **Performance Testing**:
    - Simulated slower network conditions (3G, 4G) to ensure smooth page load times and functionality under varying internet speeds.
    - Rotated screens to assess how the design adapts to both **portrait** and **landscape** orientations on different devices.
4. **Error & Console Monitoring**:
    - Monitored the **Console tab** in Developer Tools to detect any potential **JavaScript errors** or warnings that could affect page performance or user experience.

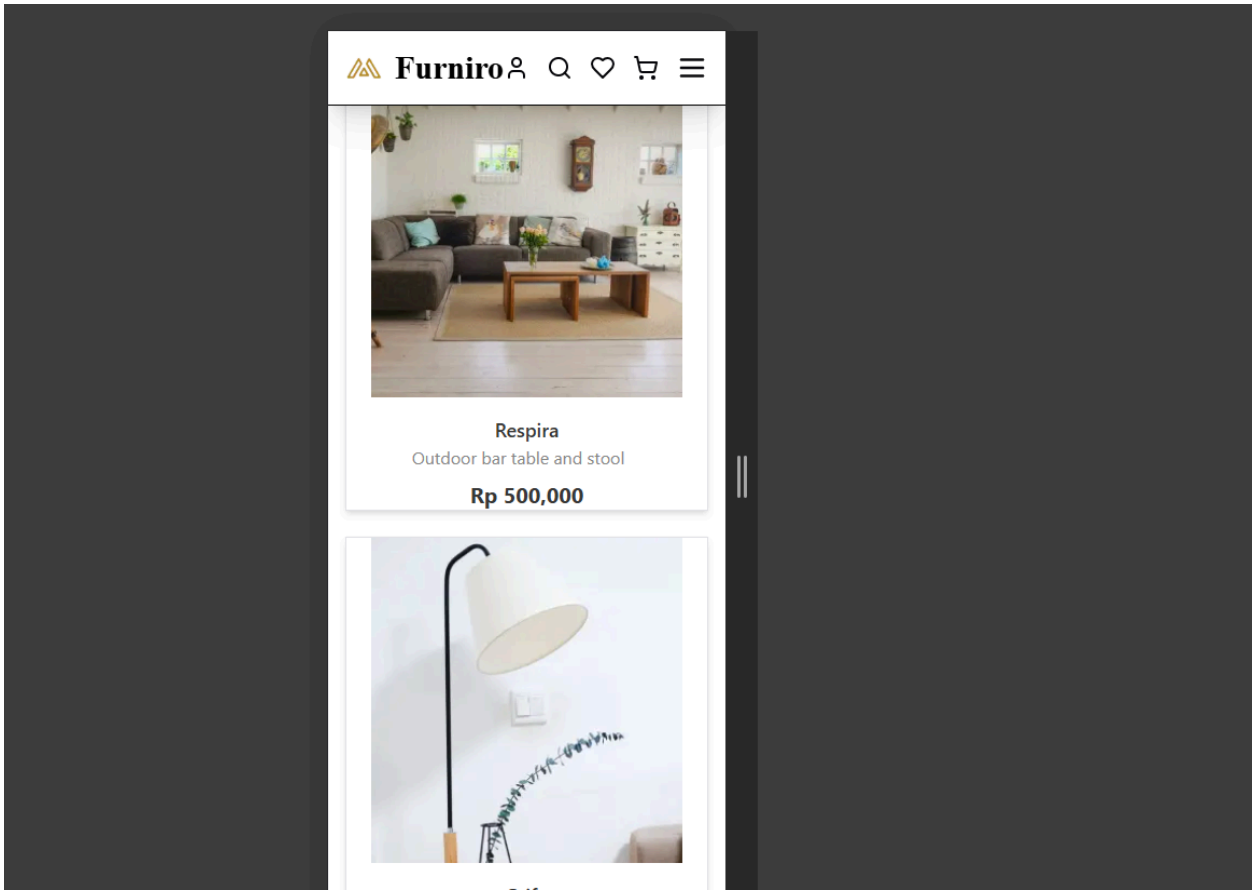
Outcome:

- **Flawless Rendering**: The website rendered seamlessly across **tablet**, **laptop**, and **mobile** screen sizes with no layout breakages or functional issues.
- **Performance**: Excellent performance across simulated network conditions, with pages loading optimally even under slower connections.
- **No Errors**: No JavaScript errors or functionality issues detected during testing, confirming stability across devices.

10) Device Testing:

- I tested the website on mobile, tablet, and laptop screens to ensure it looks great and works perfectly across all devices.
- Using **Microsoft Edge's Developer Tools**, I checked how the site adapts to different screen sizes, from small mobile phones to larger laptops and tablets.
- The site was tested in both portrait and landscape modes to ensure everything stays aligned and easy to use.
- No issues like cut-off text, broken buttons, or layout problems were found, and the site performed smoothly across all devices.





11) User Acceptance Testing (UAT)

Goal

The objective of User Acceptance Testing (UAT) is to test the product from the end-user's perspective, verify business requirements, and ensure the product is ready for final approval by checking its usability and functionality in real-world scenarios. UAT ensures the product is user-friendly and minimizes risks. This process confirms the final quality and usability of the product.

Working:

1)Simulate Real-World Usage:

- Products show correctly in my marketplace and all the product details perfectly in dynamically in my site

- Items were correctly add in cart and show correct data as user selected with multiple functionality
- Checkout page show great after the selection of products and cart page it show all the selected items of cart with its quantity total price and range customer must fill billing details as mentioned in my marketplace.
- Lastly user interact with contact page in which all the compulsory information is present for the confirmation of order! Its work Great.

2)Feedback Collection:

"I conducted comprehensive testing of my marketplace by involving friends, seniors, and mentors. Their invaluable feedback provided insightful perspectives, contributing to both positive reinforcement and constructive suggestions. The responses were overwhelmingly positive, with suggestions that further refined and enhanced the user experience, ensuring the platform's overall functionality and usability."

Testing Report(CSV Format)

TestCase Id	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Remarks				
TC001	Validate product listing page	Open product page > Verify products			Products displayed correctly		Products displayed correctly as expected	Passed	Low	No issues found	
TC002	Filter Products Functionality	Test Search > Check Output			Result according to query		Search product according to productname functional	Passsed	Low	No issues found	
TC003	Dynamic Routes	Ensure Individual Page > Load Correctly			Dynamic page must work properly		Dynamic routes work properly as expected	Passed	High	Work properly	
TC004	Test API error handling	Disconnect internet > Refresh site			Show fallback Ui with message		Show correct fallback ui error message	Passed	HIgh	Work correctly	
TC005	Operate Cart functionality	Add product to cart > Verify content			Cart update with selected products		Cart updates correctly	Passed	High	Functionality operates as expected	
TC006	Ensure Cross Browser Check	Resize browser window > check display			Layout adjust properly		Layout adjust correctly on microsoft edge and googl	Passed	Medium	Display as expected	
TC007	Responsive Designing	Inspect browser >Check response			Site show responsive		Site Show Completely Responsive	Passed	Medium	Test Successfull	
TC008	Checkout Form	Check all the fields			fields show eror on wrong info		not working correctly	Fail	Low	Week implementation	
TC009	Contact Page	Fill all the required fields for order confirmation			fields work properly		Work correctly	Passed	High	Work as expected	